CMSC 754: Lecture 20
Motion Planning

Reading: Chapt 13 in the 4M's.

Motion planning: In this lecture we will discuss applications of computational geometry to the problem of motion planning. This problem arises in robotics and in various areas where the objective is to plan the collision-free motion of a moving agent in a complex environment.

Work Space and Configuration Space: The environment in which the robot operates is called its *work space*, which consists of a domain in which the robot can move and a set of *obstacles*, which the robot must avoid. Any overlap between the robot and an obstacle is called a *collision*. We assume that the work space is static, that is, the obstacles do not move. We also assume that a complete geometric description of the work space is available to us.

For our purposes, a *robot* will be modeled by two main elements. The first is a *configuration*, which is a finite sequence of values that fully specifies the position of the robot. The second element is the robot's geometric shape description (relative to some default placement). Combined, these two elements fully define the robot's exact position and shape in space.

For example, suppose that the robot is a triangle that can translate and rotate in the plane (see Fig. 1(a)). Its configuration may be described by the (x, y) coordinates of some reference point for the robot, and an angle θ that describes its orientation. Its geometric information would include its shape (say at some canonical position), given, say, as a simple polygon. Given its geometric description and a configuration (x, y, θ), this uniquely determines the exact position $\mathcal{R}(x, y, \theta)$ of this robot in the plane. Thus, the position of the robot can be identified with a point in the robot's *configuration space*.

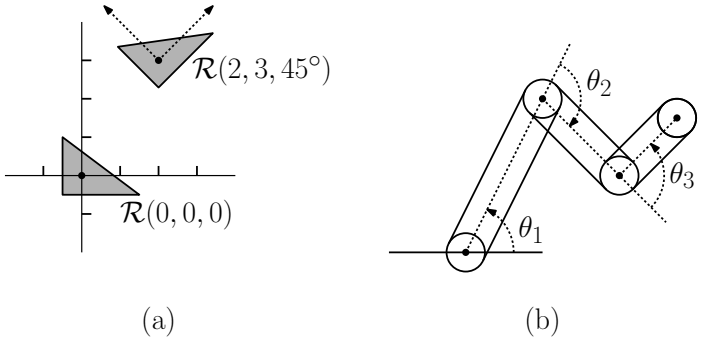


Fig. 1: Configurations of a translating and rotating robot.

A more complex example would be an *articulated arm* consisting of a set of *links*, connected to one another by a set of rotating *joints*. The configuration of such a robot might consist of a vector of joint angles $(\theta_1, \ldots, \theta_k)$ (see Fig. 1(b)). The geometric description would probably consist of a geometric representation of the links. Given a sequence of joint angles, the exact shape of the robot could be derived by combining this configuration information with its geometric description. For example, a typical 3-dimensional industrial robot has six joints, and hence its configuration can be thought of as a point in a 6-dimensional space. Why

six? Generally, there are three degrees of freedom needed to specify a location the (x, y, z) coordinates of its location in 3-space, and 3 more degrees of freedom needed to specify the direction and orientation of the robot's end manipulator. Given a point p in the robot's configuration space, let $\mathcal{R}(p)$ denote the *placement* of the robot at this configuration (see Fig. 1).

The problem of computing a collision-free path for the robot can be reduced to computing a path in the robot's configuration space. To distinguish between these, we use the term *work space* to denote the (standard Euclidean) space where the robot and obstacles reside (see Fig. 2(a)), and the *configuration space* to denote to the space in which each point corresponds to the robot's configuration (see Fig. 2(b)). Planning the motion of the robot reduces to computing a path in configuration space.

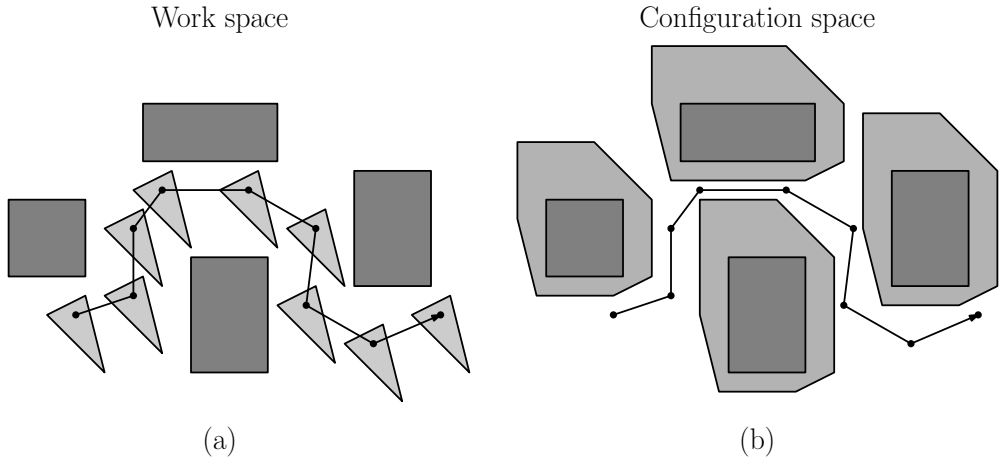


Fig. 2: (a) Translational motion in the work space amidst obstacles and (b) motion in the configuration space amidst collision obstacles.

A configuration that results in the robot to intersecting with one or more of the obstacles is called a *forbidden configuration*. The set of all forbidden configurations is denoted $C_{\text{forb}}(\mathcal{R}, S)$. All other placements are called *free configurations*, and the set of these configurations is denoted $C_{\text{free}}(\mathcal{R}, S)$, or *free space*.

Now consider the *motion planning* problem in robotics. Given a robot $\mathcal{R}$, an work space S, and initial configuration s and final configuration t (both points in the robot's free configuration space), determine (if possible) a way to move the robot from one configuration to the other without intersecting any of the obstacles. This reduces to the problem of determining whether there is a path from s to t that lies entirely within the robot's free configuration space. Thus, we map the task of computing a robot's motion to the problem of finding a path for a single point through a collection of obstacles.

Configuration Obstacles: While motion occurs in the robot's work space, we perform motion planning in the robot's configuration space. For example, given a 2-dimensional robot that can translate and rotate amidst 2-dimensional polygonal objects, we determine collisions in the 2-dimensional space. But our motion plan is a path in 3-dimensional (x, y, θ) space, where (x, y) indicate the robot's translation and θ gives its rotation (about the reference point).

In order to determine which configurations are free and forbidden, we need to determine the meaning of an obstacle in configuration space. Let's suppose that the work space is d-dimensional and the configuration space is k-dimensional. Given an obstacle $P \subset \mathbb{R}^d$, its *configuration obstacle* or (or *C-obstacles* for short) is defined to be

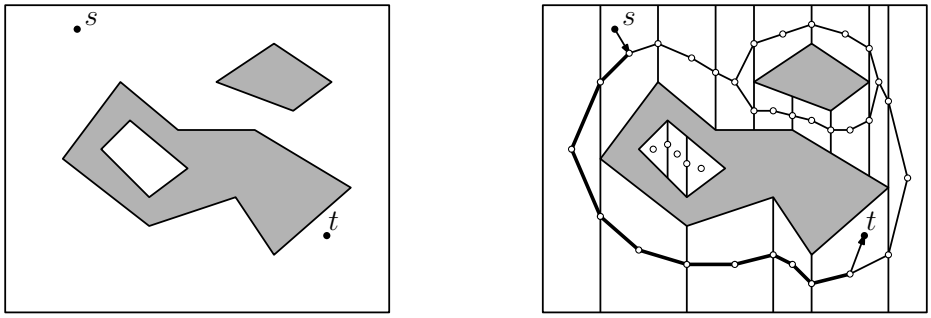
$$\mathcal{C}_{\mathcal{R}}(P) \;=\; \{p \in \mathbb{R}^k : \mathcal{R}(p) \cap P \neq \emptyset\}.$$

(Note that the configuration p is in $\mathbb{R}^k$, but the determination of collision between $\mathcal{R}(p)$ and P is done in the work space $\mathbb{R}^d$.)

Configuration spaces are typically higher dimensional than the work space, and configuration obstacles can be quite complex. When rotation is involved, they are bounded by curved surfaces. The simplest case to visualize is that of translating a convex polygonal robot in the plane amidst a collection of polygonal obstacles, since both the work space and configuration space are two dimensional. Consider a reference point placed in the center of the robot. The process of mapping to configuration space involves replacing the robot with a single point (its reference point) and "growing" the obstacles by a compensating amount (see Fig. 2(b)).

For the rest of the lecture we will consider a very simple case of a convex polygonal robot that is translating among a convex set of obstacles. Even this very simple problem has a number of interesting algorithmic issues.

Planning the Motion of a Point Robot: As mentioned above, we can reduce complex motion planning problems to the problem of planning the motion of a point in free configuration space. First we will consider the question of how to plan the motion of a point amidst a set of obstacles, and then we will consider the question of how to construct configuration spaces.



(a) (b)

Fig. 3: Simple point motion planning through road maps.

Let us start with a very simple case in which the configuration space is 2-dimensional and the objects are simple polygons, possibly with holes (see Fig. 3(a)). To determine whether there is a path from one point s to another point t of free configuration space, we can subdivide free space into simple convex regions. In the plane, we already know of one way to do this by computing a trapezoidal map. We construct a trapezoidal map for all of the line segments bounding the obstacles, then throw away any trapezoids that lie in the forbidden space (see Fig. 3(b)). We also assume that we have a point location data structure for the trapezoidal map.

Next, we create a planar graph, called a *road map*, based on this subdivision. To do this we create a vertex in the center of each trapezoid and a vertex at the midpoint of each vertical edge. We create edges joining each center vertex to the vertices on its (at most four) edges.

Now to answer the motion planning problem, we assume we are given the start point s and destination point t. We locate the trapezoids containing these two points, and connect them to the corresponding center vertices. We can join them by a straight line segment, because the cells of the subdivision are convex. Then we determine whether there is a path in the road map graph between these two vertices, say by breadth-first search. Note that this will not necessarily produce the shortest path, but if there is a path from one position to the other, it will find it.

Practical Considerations: While the trapezoidal map approach guarantees correctness, it is rather limited. If the configuration space is 2-dimensional, but the configuration obstacles have curved boundaries, we can easily extend the trapezoidal map approach, but we will generally need to insert walls at points of vertical tangency.

Higher-dimensional spaces pose a much bigger problem (especially when combined with curved boundaries). There do exist subdivision methods (one is called the *Collins cylindrical algebraic decomposition*, which can be viewed as a generalization of the trapezoidal map to higher dimensions and curved surfaces), but such subdivisions often can have high combinatorial complexity. Most practical road map-based approaches dispense with computing the subdivision, and instead simply generate a large random sample of points in free space. The problem is that if no path is found, who is to blame? Is there really no path, or did we simply fail to sample enough points? The problem is most extreme when the robot needs to navigate through a very narrow passage.

Another widely used heuristic is called the *rapidly-exploring random tree* (RRT). These trees provide a practical approach to sampling the configuration space and building a tree-based road map. While this method has good practical value, it can also fail when tight squeezes are necessary.

C-Obstacles via Minkowski Sums: Let us consider how to build a configuration space for a set of polygonal obstacles. We consider the simplest case of translating a convex polygonal robot amidst a collection of convex obstacles. If the obstacles are not convex, then we may subdivide them into convex pieces. One way to visualize $\mathcal{C}_{\mathcal{R}}(P)$ is to imagine "scraping" $\mathcal{R}$ along the boundary of P and seeing the region traced out by $\mathcal{R}$'s reference point (see Fig. 4(a)).

Given $\mathcal{R}$ and P, how do we compute the configuration obstacle $\mathcal{C}_{\mathcal{R}}(P)$? To do this, we first introduce the notion of a *Minkowski sum*. Let us think of points in the plane as vectors. Given any two sets P and Q in the plane, define their *Minkowski sum* to be the set of all pairwise sums of points taken from each set (see Fig. 4(b)), that is,

$$P \oplus Q \;=\; \{p + q \;:\; p \in P,\; q \in Q\}.$$

Also, define $-S = \{-p \;:\; p \in S\}$. (Intuitively, we are reflecting S through the origin.) We introduce the shorthand notation $\mathcal{R} \oplus p$ to denote $\mathcal{R} \oplus \{p\}$. Observe that the *translate* of $\mathcal{R}$ by vector p is $\mathcal{R}(p) = \mathcal{R} \oplus p$. The relevance of Minkowski sums to C-obstacles is given in the following claim.

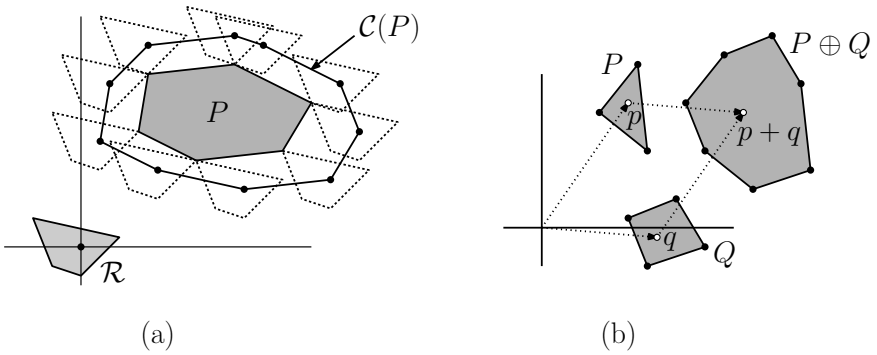


Fig. 4: Minkowski sum of two polygons.

Lemma: Given a translating robot $\mathcal{R}$ and an obstacle P, $\mathcal{C}_{\mathcal{R}}(P) = P \oplus (-\mathcal{R})$ (see Fig. 5).

Proof: Consider any translation vector t. Observe that $t \in \mathcal{C}_{\mathcal{R}}(P)$ iff $\mathcal{R}(t)$ intersects P, which is true iff there exist $r \in \mathcal{R}$ and $p \in P$ such that $p = r + t$ (see Fig. 5(a)), which is true iff there exist $-r \in -\mathcal{R}$ and $p \in P$ such that $t = p + (-r)$ (see Fig. 5(b)), which is equivalent to saying that $t \in P \oplus (-\mathcal{R})$. Therefore, $t \in \mathcal{C}_{\mathcal{R}}(P)$ iff $t \in P \oplus (-\mathcal{R})$, which means that $\mathcal{C}_{\mathcal{R}}(P) = P \oplus (-\mathcal{R})$, as desired.

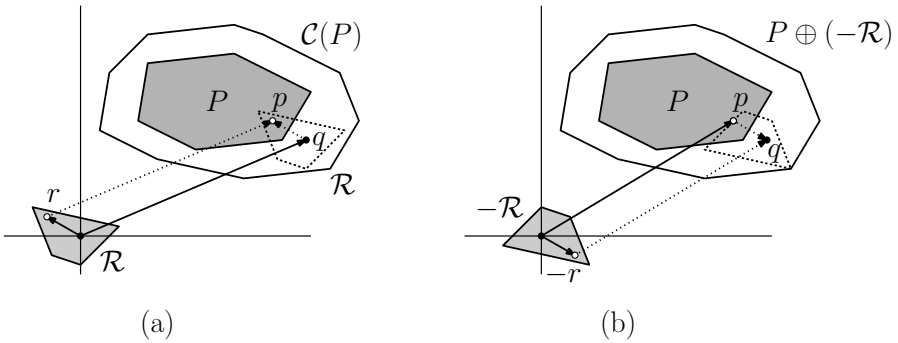


Fig. 5: Configuration obstacles and Minkowski sums.

It is an easy matter to compute $-\mathcal{R}$ in linear time (by simply negating all of its vertices) the problem of computing the C-obstacle $\mathcal{C}_{\mathcal{R}}(P)$ reduces to the problem of computing a Minkowski sum of two convex polygons. We'll show next that this can be done in $O(m+n)$ time, where m is the number of vertices in $\mathcal{R}$ and n is the number of vertices in P.

Note that the above proof made no use of the convexity of $\mathcal{R}$ or P. It works for any shapes and in any dimension. However, computation of the Minkowski sums is most efficient for convex polygons.

Computing the Minkowski Sum of Convex Polygons: Let's consider how to compute $P \oplus \mathcal{R}$ for two convex polygons P and $\mathcal{R}$, having m and n vertices, respectively. The algorithm is based on the following observation. Given a vector u, We say that a point p is *extreme* in direction u if it maximizes the dot product $p \cdot u$ (equivalently, a support line perpendicular to

u passes through p with the outward normal u). The following observation is easy to prove by the linearity of the dot product.

Observation: Given two polygons P and $\mathcal{R}$, the set of extreme points of $P \oplus \mathcal{R}$ in direction u is the set of sums of points p and r that are extreme in direction u for P and $\mathcal{R}$, respectively.

This observation motivates an algorithm for computing $P \oplus \mathcal{R}$. We perform an angular sweep by sweeping a unit vector u counterclockwise around a circle. As u rotates, it is an easy matter to maintain the vertex or edge of P and $\mathcal{R}$ that is extreme in this direction. Whenever u is perpendicular to an edge of either P or $\mathcal{R}$, we add this edge to the vertex of the other polygon. The algorithm is given in the text, and is illustrated in the figure below. The technique of applying one or more angular sweeps to a convex polygon is called the method of *rotating calipers*.

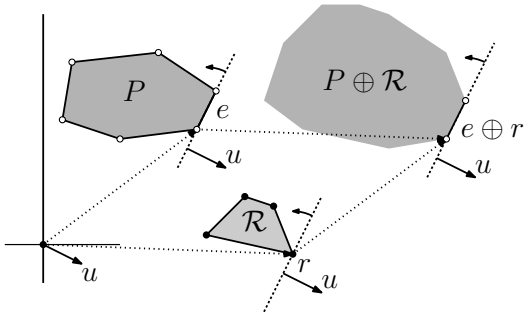


Fig. 6: Computing Minkowski sums.

Assuming P and $\mathcal{R}$ are convex, observe that each edge of P and each edge of $\mathcal{R}$ contributes exactly one edge to $P \oplus \mathcal{R}$. (If two edges are parallel and on the same side of the polygons, then these edges will be combined into one edge, which is as long as their sum.) Thus we have the following.

Claim: Given two convex polygons, P and $\mathcal{R}$, with n and m edges respectively, their Minkowski sum $P \oplus \mathcal{R}$ can be computed in $O(n + m)$ time, and consist of at most $n + m$ edges.

Complexity of Minkowski Sums: We have shown that free space for a translating robot is the complement of a union of C-obstacles $\mathcal{C}_{\mathcal{R}}(P)_i$, each of which is a Minkowski sum of the form $P_i \oplus \mathcal{R}$, where P_i ranges over all the obstacles in the environment. If P_i and $\mathcal{R}$ are polygons, then the resulting region will be a union of polygons. How complex might this union be, that is, how many edges and vertices might it have?

To begin with, let's see just how bad things might be. Suppose you are given a robot $\mathcal{R}$ with m sides and a set of work-space obstacle P with n sides. How many sides might the Minkowski sum $P \oplus \mathcal{R}$ have in the worst case? $O(n+m)$? $O(nm)$, even more? The complexity generally depends on what special properties if any P and $\mathcal{R}$ have.

Nonconvex Robot and Nonconvex Obstacles: Suppose that both $\mathcal{R}$ and P are (possibly nonconvex) simple polygons. Let m be the number of sides of $\mathcal{R}$ and n be the number of sides of P. How many sides might there be in the Minkowski sum $P \oplus \mathcal{R}$ in the worst case? We can derive a quick upper bound as follows. First observe that if we triangulate P, we can break it into the union of at most $n-2$ triangles. That is:

$$P = \bigcup_{i=1}^{n-2} T_i \quad \text{and} \quad \mathcal{R} = \bigcup_{j=1}^{m-2} S_j.$$

It follows that

$$P \oplus \mathcal{R} = \bigcup_{i=1}^{n-2} \bigcup_{j=1}^{m-2} (T_i \oplus S_j).$$

Thus, the Minkowski sum is the union of $O(nm)$ polygons, each of constant complexity. Thus, there are $O(nm)$ sides in all of these polygons. The arrangement of all of these line segments can have at most $O(n^2m^2)$ intersection points (if each side intersects with each other), and hence this is an upper bound on the number of vertices in the final result.

Could the complexity really be this high? Yes it could. Consider the two polygons in Fig. 7(a). Suppose that P and $\mathcal{R}$ have m and n "teeth", respectively. For each of independent choice of two teeth of P (one from the top and one from the side), and two gaps from $\mathcal{R}$ (one from the top and one from the side), there is a valid placement where these teeth fit within these gaps (see the arrows in Fig. 7(a)). However, as can be seen from the figure, it is impossible to move from one of these to another by translation without causing a collision. It follows that there are $\Omega(n^2m^2)$ connected components of the free configuration space, or equivalently in $P \oplus -\mathcal{R}$ (see Fig. 7(b)).

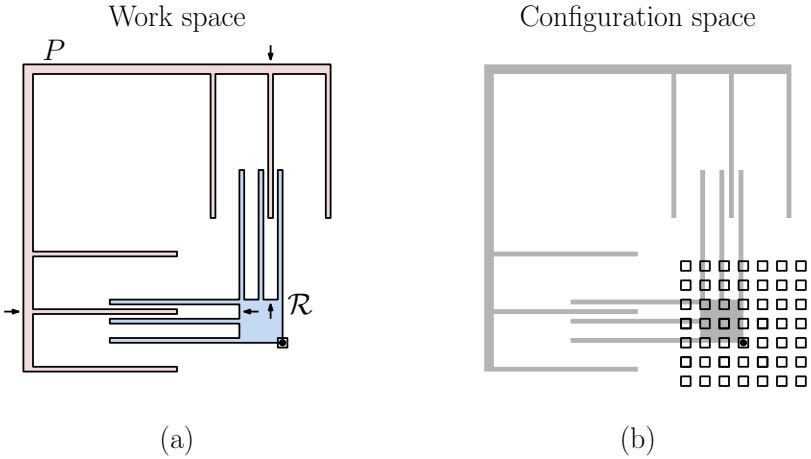


Fig. 7: Minkowski sum (simple-simple) of $O(n^2m^2)$ complexity.

You might protest that this example is not fair. While it is true that there are many components in the Minkowski sum, motion planning takes place within a single connected component of free space, and therefore the quantity that is really of interest is the (worst-case) combinatorial complexity of any *single* connected component of free space. (In the example above,

all the components were of constant complexity.) This quantity is complicated to bound for general shapes, but later we will show that it can be bounded for convex shapes.

As a final observation, notice that the upper bound holds even if P (and $\mathcal{R}$ for that matter) is not a single simple polygon, but any union of n triangles.

Convex Robot and Nonconvex Obstacles: We have seen that the worst-case complexity of the Minkowski sum might range from $O(n+m)$ to as high as $O(n^2m^2)$, which is quite a gap. Let us consider an intermediate but realistic situation. Suppose that we assume that P is an arbitrary n-sided simple polygon, and R is a convex m-sided polygon. Typically m is much smaller than n. What is the combinatorial complexity of $P \oplus \mathcal{R}$ in the worst case? As before we can observe that P can be decomposed into the union of $n-2$ triangles T_i, implying that

$$P \oplus \mathcal{R} = \bigcup_{i=1}^{n-2} (T_i \oplus \mathcal{R}).$$

Each Minkowski sum in the union is of complexity $m+3$. So the question is how many sides might there be in the union of $O(n)$ convex polygons each with $O(m)$ sides? We could derive a bound on this quantity, but it will give a rather poor bound on the worst-case complexity. To see why, consider the limiting case of $m = 3$. We have the union of n convex objects, each of complexity $O(1)$. This could have complexity as high as $\Omega(n^2)$, as seen by generating a criss-crossing pattern of very skinny triangles. But, if you try to construct such a counter example, you won't be able to do it.

To see why such a counterexample is impossible, suppose that you start with nonintersecting triangles, and then take the Minkowski sum with some convex polygon. The claim is that it is impossible to generate this sort of criss-cross arrangement. So how complex an arrangement can you construct? We will show the following later in the lecture.

Theorem: Let $\mathcal{R}$ be a convex m-gon and P and simple n-gon, then the Minkowski sum $P \oplus \mathcal{R}$ has total complexity $O(nm)$.

Is $O(nm)$ an attainable bound? The idea is to go back to our analogy of "scraping" $\mathcal{R}$ around the boundary of P. Can we arrange P such that most of the edges of $\mathcal{R}$ scrape over most of the n vertices of P? Suppose that $\mathcal{R}$ is a regular convex polygon with m sides, and that P has a comb-like structure where the teeth of the comb are separated by a distance at least as large as the diameter of $\mathcal{R}$ (see Fig. 8(a)). In this case $\mathcal{R}$ will have many sides scrape across each of the pointy ends of the teeth, implying that the final Minkowski sum will have total complexity $\Omega(nm)$ (see Fig. 8(b)).

The Union of Pseudodisks: Consider a translating robot given as an m-sided convex polygon and a collection of polygonal obstacles having a total of n vertices. We may assume that the polygonal obstacles have been triangulated into at most n triangles, and so, without any loss of generality, let us consider an instance of an m-sided robot translating among a set of n triangles. As argued earlier, each C-obstacle has $O(3+m) = O(m)$ sides, for a total of $O(nm)$ line segments. A naive analysis suggests that this many line segments might generate as many as $O(n^2m^2)$ intersections, and so the complexity of the free space can be no larger. However,

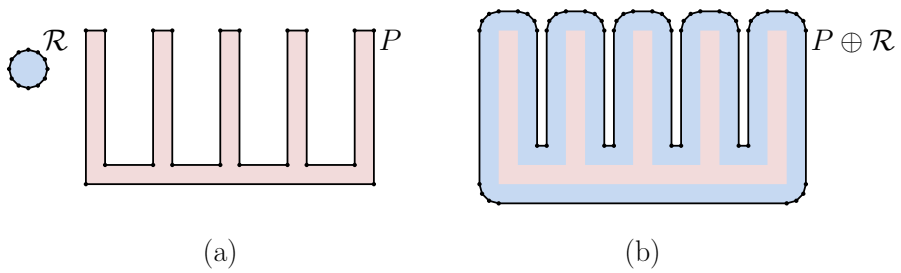


Fig. 8: Minkowski sum (simple-convex) of $O(nm)$ complexity.

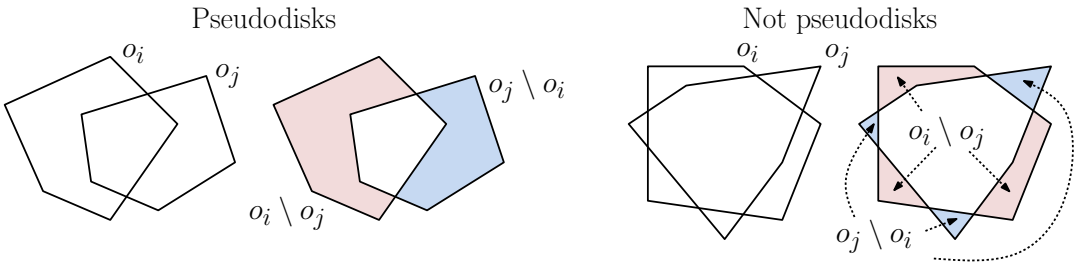


Fig. 9: Pseudodisks.

we assert that the complexity of the space will be much smaller, in fact its complexity will be $O(nm)$.

To show that $O(nm)$ is an upper bound, we need some way of extracting the special geometric structure of the union of Minkowski sums. Recall that we are computing the union of $T_i \oplus \mathcal{R}$, where the T_i's have disjoint interiors. A set of convex objects $\{o_1, \ldots, o_n\}$ is called a *collection of pseudodisks* if for any two distinct objects o_i and o_j both of the set-theoretic differences $o_i \setminus o_j$ and $o_j \setminus o_i$ are connected (see Fig. 9). If this is violated for any two objects, we say that these two objects have a *crossing intersection*. Note that the pseudodisk property is not a property of a single object, but a property that holds for a set of objects.

Lemma 1: Given a set convex objects $T_1, \ldots, T_n$ with disjoint interiors, and convex $\mathcal{R}$, then

$$\{T_i \oplus \mathcal{R} \mid 1 \leq i \leq n\}$$

is a collection of pseudodisks (see Fig. 10).

Proof: Consider two polygons T_1 and T_2 with disjoint interiors. We want to show that $T_1 \oplus \mathcal{R}$ and $T_2 \oplus \mathcal{R}$ do not have a crossing intersection. Given any directional unit vector u, the *most extreme* point of $\mathcal{R}$ in direction u is the point $r \in \mathcal{R}$ that maximizes the dot product $(u \cdot r)$. (Recall that we treat the "points" of the polygons as if they were vectors.) The point of $T_1 \oplus \mathcal{R}$ that is most extreme in direction u is the sum of the points t and r that are most extreme for T_1 and $\mathcal{R}$, respectively.

Given two convex polygons T_1 and T_2 with disjoint interiors, they define two outer tangents, as shown in the figure below. Let u_1 and u_2 be the outward pointing perpendicular vectors for these tangents. Because these polygons do not intersect, it follows easily that as the directional vector rotates from u_1 to u_2, T_1 will be the more extreme polygon, and from u_2 to u_1 T_2 will be the more extreme (see Fig. 11).

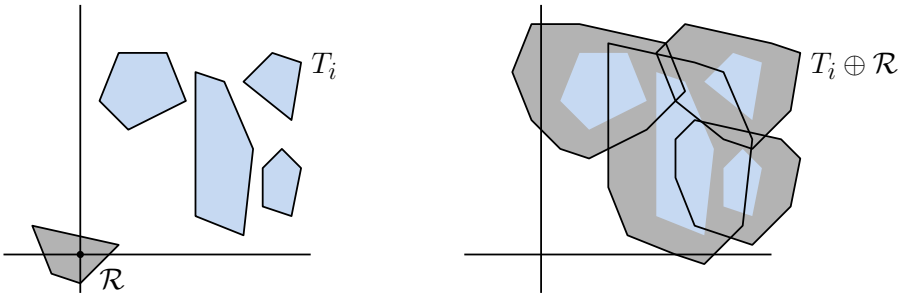


Fig. 10: Lemma 1.

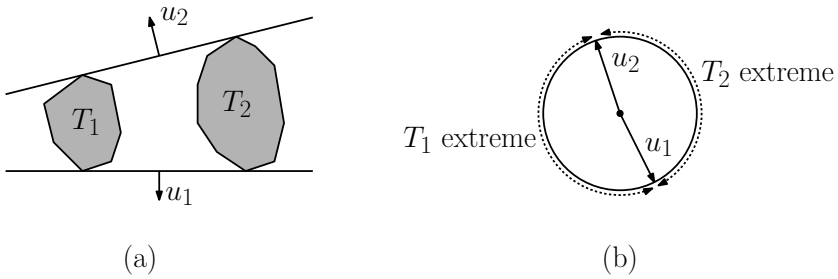


Fig. 11: Alternation of extremes.

Now, if to the contrary $T_1 \oplus \mathcal{R}$ and $T_2 \oplus \mathcal{R}$ had a crossing intersection, then observe that we can find points p_1 p_2, p_3, and p_4, in cyclic order around the boundary of the convex hull of $(T_1 \oplus \mathcal{R}) \cup (T_2 \oplus \mathcal{R})$ such that $p_1, p_3 \in T_1 \oplus \mathcal{R}$ and $p_2, p_4 \in T_2 \oplus \mathcal{R}$. First consider p_1. Because it is on the convex hull, consider the direction u_1 perpendicular to the supporting line here. Let r, t_1, and t_2 be the extreme points of $\mathcal{R}$, T_1 and T_2 in direction u_1, respectively. From our basic fact about Minkowski sums we have

$$p_1 = r + t_1 \qquad p_2 = r + t_2.$$

Since p_1 is on the convex hull, it follows that t_1 is more extreme than t_2 in direction u_1, that is, T_1 is more extreme than T_2 in direction u_1. By applying this same argument, we find that T_1 is more extreme than T_2 in directions u_1 and u_3, but that T_2 is more extreme than T_1 in directions u_2 and u_4. But this is impossible, since from the observation above, there can be at most one alternation in extreme points for nonintersecting convex polygons (see Fig. 12).

Lemma 2: Given a collection of polygonal pseudodisks, with a total of n vertices, the complexity of their union is $O(n)$.

Proof: This is a rather cute combinatorial lemma. We are given some collection of polygonal pseudodisks, and told that altogether they have n vertices. We claim that their entire union has complexity $O(n)$. (Recall that in general the union of n convex polygons can have complexity $O(n^2)$, by criss-crossing.) The proof is based on a clever charging

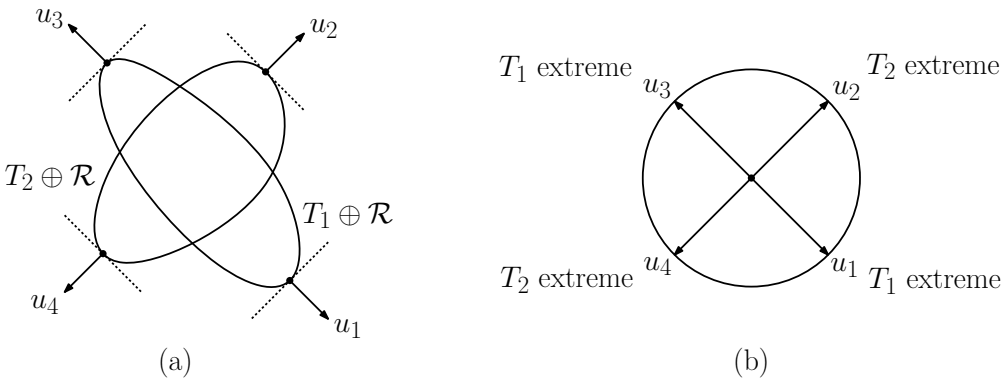


Fig. 12: Proof of Lemma 1.

scheme. Each vertex in the union will be charged to a vertex among the original pseudodisks, such that no vertex is charged more than twice. This will imply that the total complexity is at most $2n$.

There are two types of vertices that may appear on the boundary. The first are vertices from the original polygons that appear on the union. There can be at most n such vertices, and each is charged to itself. The more troublesome vertices are those that arise when two edges of two pseudodisks intersect each other. Suppose that two edges e_1 and e_2 of pseudodisks P_1 and P_2 intersect along the union. Follow edge e_1 into the interior of the pseudodisk e_2. Two things might happen. First, we might hit the endpoint v of this e_1 before leaving the interior P_2. In this case, charge the intersection to v (see Fig. 13(a)). Note that v can be assessed at most two such charges, one from either incident edge. If e_1 passes all the way through P_2 before coming to the endpoint, then try to do the same with edge e_2. Again, if it hits its endpoint before coming out of P_1, then charge to this endpoint (see Fig. 13(b)).

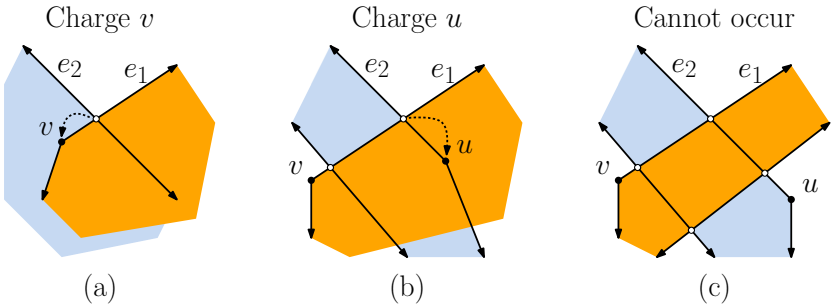


Fig. 13: Proof of Lemma 2.

But what do we do if both e_1 shoots straight through P_2 and e_2 shoots straight through P_1? Now we have no vertex to charge. This is okay, because the pseudodisk property implies that this cannot happen. If both edges shoot completely through, then the two polygons must have a crossing intersection (see Fig. 13(c)).

Recall that in our application of this lemma, we have n C-obstacles, each of which has at

most $m + 3$ vertices, for a total input complexity of $O(nm)$. Since they are all pseudodisks, it follows from Lemma 2 that the total complexity of the free space is $O(nm)$.